Laboratorul 3

Procese, Fire si Memoria Virtuala:

Un proces reprezinta multimea de resurse alocate pentru executia unui program. Abstractizand destul de mult, un proces este format din :

- Memorie virtuala
- Un program care se executa
- O lista de handle-uri catre diferite resurse (fisiere, chei de registry, obiecte de sincronizare, etc.)
- Access token
- Un identificator unic (process id = PID)
- Cel putin un fir de executie

Memorie Virtuala:

Daca datele pentru fiecare proces ar fi scrise direct in memoria fiziaca, fara niciun fel de control, atunci ar exista posibilitatea ca un proces sa scrie accidental peste zona de memorie ce apartine altui proces. Acest lucru se intampla la sistmeul de operare MS-DOS si a reprezentat unul din punctele principale care trebuiau imbunatatite atunci cand s-a trecut la sistemul de operare Windows.

In loc sa scrie direct in memoria fizica, fiecare proces scrie intr-o zona proprie numita memorie virtuala. Este cate o zona de memorie virtuala pentru fiecare proces. Orice proces, va considera ca are la dispozitie zona maxima de memorie specifica arhitecturii sistemului si poate scrie in orice zona (atat timp cat este alocata si are drepturi de scriere pe zona respectiva).

Un sistem central va face traducerea intre o adresa virtuala a unui proces si adresa fizica din RAM. Practic, doua procese pot scrie in aceiasi adresa virtuala, dar sistemul central va avea grija sa aloce adrese fizice diferite pentru fiecare proces.

In felul acesta, un proces nu poate scrie accidental in memoria altui proces. Mai mult, un proces va functiona la fel indiferent de cata memorie are instalata sistemul respectiv. Sistemul de operare va stii sa faca swapping pe disk atunci cand ramane fara memorie libera, dar un proces nu trebuie sa stie acest lucru.

Fiecare proces va incarca mai multe module pentru a se putea executa.Un modul reprezinta fie un fisier executabil, fie un fisier de tip librarie care exporta unele functionalitati. Aceste module sunt incarcate tot in memoria virtuala la o adresa ceruta de modul sau relocate daca acea adresa este deja ocupata. Tinand cont ca multe din module vor fi incarcate de majoritatea proceselor (ex: kernel32.dll, user32.dll, advapi32.dll) sistemul central de management a memoriei va stii sa aloce memorie o singura data pentru fiecare modul, iar zona de memorie virtuala pentru fiecare proces care foloseste acelasi modul va indica spre aceiasi adresa in memoria fizica.

Access Token

Atunci cand se logheaza un utilizator se creaza si un access –token prin care sa fie identificat utilizatorul. De fiecare data cand un proces este creat, el va mostenii tokenul parintelui. Astfel, toate procesele din sesiunea unui utilizator vor avea acelasi token (sunt exceptii de la regula in care un proces poate folosi alt token)

Deci access token-ul reprezinta multimea de informatii ce identifica userul in contextul caruia se executa procesul , privilegiile pe care le are procesul respectiv pe sistem precum si drepturile asupra procesului.

Principalele elemente care se gasesc intr-un access-token sunt:

- Un numar de securitate (SID) prin care se identifica utilizatorul
- Un numar de securitate (DIS) prin care se identifica grupul
- O lista de privilegii detinute de user sau de grup
- O lista de drepturi prin care se identifica ce actiuni au dreptul alti utilizatori ai sistemului sa execute asupra acestui proces (de ex. Sa termine procesul)

Diferenta intre drept de securitate si privilegiu este data de nivelul la care se aplica. Dreptul de securitate se refera la ce poate un proces (utilizator) sa faca asupra unui obiect speicific (ex:sa deschida un fisier pentru citire, sa opreasca un proces, sa scrie in memoria unui proces, etc.) (Drepturile de securitate vor fi studiate intr-un laborator viitor). Privilegiul se refera la actiunile pe care un proces poate sa le faca la nivel de sistem (ex: sa inchida sistmeul, sa faca debug la sistem, sa faca backup, etc.).

Fire de executie

Firul de executie reprezinta o entitate a unui proces, pe care sistemul de operare o programeaza la executia pe unul din procesoare. Un fir contine urmatoarele componente:

- Un set de registri (eax, ebx, eip ...)
- Doua stive (una pentru kernel , una pentru user mode)
- O zona din memoria procesului care va fi folosita pentru a tine diferite date specifice (TLS = thread local storage)
- Un identificator unic (TID = thread id)
- Un security token (asemanator ca in cazul procesului)

Primele 3 elemente (registrii, stivele si TLS-ul) formeaza thread-contextul. De fiecare data cand se un fir de executie paraseste procesorul, thread context-ul este salvat si se incarca alt fir de executie (alt thread-context).

Pentru mai multe detalii legate de procese si fire, consultati pagina cursului.

Lucru cu procese si fire de executie

Pentru **enumerarea** proceselor exista doua modalitati

1. Prin apelarea functiei CreateToolhelp32Snapshot

```
HANDLE WINAPI CreateToolhelp32Snapshot(
  _In_  DWORD dwFlags,
  _In_  DWORD th32ProcessID
);
```

Prin apelarea acestei functii, se va creea un shapshot la procese, module sau fire de executie, in functie de ce parametru este dat la dwFlags.

dwFlags	Indica ce vrem sa contina snapshotul. TH32CS_SNAPMODULE = pentru listarea modulelor TH32CS_SNAPPROCESS = pentru listarea proceselor TH32CS_SNAPTHREAD = pentru listarea firelor de executie
th32ProcessID	Pentru listarea proceselor si listarea firelor de executie este ignorat. Pentru listarea modulelor sau (a zonelor de heap) indica procesul pentru care se doreste listarea

Cand se cere listarea firelor de executie, se vor lista toate firele ce se afla pe sistemul curent, nu doar cele care apartin unui proces. Acest lucru se intampla pentru ca firele sunt obiecte separate pe sistem. Ele sunt asociate unui proces, dar la programare la procesor sunt considerate toate firele din sistem. Asadar, din acest motiv, parametrul th32ProcessId nu-si are sensul cand functia este apelata cu flagul TH32CS_ SNAPTHREAD.

Daca functia reuseste va returna un handle catre snapshot.
Parcurgerea snapshat-ului se face cu urmatoarele functii:
Process32First si Process32Next pentru listarea proceselor
Thread32First si Thread32Next pentru listarea firelor de executie
Module32First si Module32Next pentru listarea modulelor

Metoda de apel al acestor functii este asemanatoare cu metoda folosita pentru a parcurge fisiere (FindFirstFile si FindNextFile). Fiecare din aceste functii are2 parametrii si vor primi ca prim parametru snapshot-ul creat prin apelul functiei CreateToolhelp32Snapshot. De fiecare data cand se apeleaza una din functiile Process32Next, Thread32Next si Module32Next vor fi obtinute informatii despre urmatorul obiect din lista (unde obeict este fie un proces, fie un fir, fie un modul). Aceste informatii vor fi puse in al doilea parametru. Deci, al doilea parametru reprezinta un pointer catre o structura specifica unde vor fi returnate informatiile.

Mai jos este un exemplu de utilizare a functiilor Process32First si Process32Next pentru listarea unor procese. Observati ca inainte de apelarea functiei Process32First, elementul dwSize din structura PROCESEENTRY32, a fost initializat cu dimensiunea structurii. Acest lucru trebuie facut si pentru functiile Thread32First si Module32First.

```
int main()
{
	HANDLE hProcess;
	PROCESSENTRY32 pe32;

	//cer un snapshot la procese
	hProcessSnap = CreateToolhelp32Snapshot( TH32CS_SNAPPROCESS, 0 );

	if( hProcessSnap == INVALID_HANDLE_VALUE )
	{
		printf("CreateToolhelp32Snapshot failed.err = %d \n",GetLastError());
		return( -1 );
	}

	//initializez dwSize cu dimensiunea structurii.
	pe32.dwSize = sizeof( PROCESSENTRY32 );

	//obtin informatii despre primul proces
	if( !Process32First( hProcessSnap, &pe32 ) )
	{
		printf("Process32First failed. err = %d \n", GetLastError() );
		CloseHandle( hProcessSnap );     //inchidem snapshot-ul
		return( -1 );
	}

	do
	{
		//afisez pid-ul si executabilul
		printf("Process [%d]:  %s \n",pe32.th32ProcessID,pe32.szExeFile );
	}
	while(Process32Next(hProcessSnap, &pe32 )); //trec la urmatorul proces

	//inched handle-ul catre snapshot
	CloseHandle(hProcess);

	return 0;
}
```

2. Prin apelarea functiei EnumProcesses. Cu aceasta functie se pot afla doar pid-urile proceselor, fara alte informatii ca in cazul functiei CreateToolhelp32Snapshot

```
BOOL WINAPI EnumProcesses(
  _Out_  DWORD *pProcessIds,
  _In_   DWORD cb,
  _Out_  DWORD *pBytesReturned
);
```

pProcessIds	Reprezinta un vector in care va fi pus pe fiecare pozitie id-ul unui proces
Cb	Dimensiunea in bytes a vectorului
pBytesReturned	Cati bytes au fost scrisi in vector. Pentru a afla numarul de procese, se imparte acest numar la 4

Pentru a lucra cu un proces anume (de ex. Obtinerea de informatii despre un proces dat pid-ul, sau pentru terminarea procesului) trebuie intai obtinut intai un handle la proces. Acest lucru se face prin intermediul functiei OpenProcess.

```
HANDLE WINAPI OpenProcess(
  _In_  DWORD dwDesiredAccess,
  _In_  BOOL bInheritHandle,
  _In_  DWORD dwProcessId
);
```

dwDesiredAccess	Drepturile care indica ce operatii urmeaza sa facem asupra procesului. Ex: PROCESS_TERMINATE, PROCESS_QUERY_INFORMATION, PROCESS_ALL_ACCESS
Cb	Daca alte procese, create de acesta sa mosteneasca acest handle
dwProcessId	Id-ul procesului la care dorim handle

Daca Handle-ul a fost obtinut, atunci se pot face operatiile care au fost cerute in parametrul dwDesiredAccess.

Functii utile sunt: GetModuleBaseName, ProcessTerminate.

Pentru a crea un proces, se poate apela functia CreateProcess

```
BOOL WINAPI CreateProcess(
  _In_opt_     LPCTSTR lpApplicationName,
  _Inout_opt_  LPTSTR lpCommandLine,
  _In_opt_     LPSECURITY_ATTRIBUTES lpProcessAttributes,
  _In_opt_     LPSECURITY_ATTRIBUTES lpThreadAttributes,
  _In_         BOOL bInheritHandles,
  _In_         DWORD dwCreationFlags,
  _In_opt_     LPVOID lpEnvironment,
  _In_opt_     LPCTSTR lpCurrentDirectory,
  _In_         LPSTARTUPINFO lpStartupInfo,
  _Out_        LPPROCESS_INFORMATION lpProcessInformation
);
```

lpApplicationName	Path catre executabil
lpCommandLine	Parametrii pentru executabil. Daca lpApplicationName este NULL, atunci path-ul trebuie dat in commandline

lpProcessAttributes	Atributele de securitate pentru proces (poate fi NULL)
lpThreadAttributes	Atributele de securitate pentru firul principal (paote fi NULL)
bInheritHandles	Daca sa mosteneasca handle-uri de la procesul curent. Deobicei este FALSE.
dwCreationFlags	Deobicei este NULL. Specifica diferita flaguri despre crearea procesului. De ex, poate fi creat un proces suspended (CREATE_SUSPENDED) sau procesul creat poate avea propria consola in loc sa scrie output-ul in aceiasi consoal (CREATE_NEW_CONSOLE)
lpEnvironment	Setari specifice PEB-ului. Deobicei 0
lpCurrentDirectory	Directorul curent pentru noul Proces. Deobicei 0 (directorul curent)
lpStartupInfo	
lpProcessInformation	

Crearea si lucru cu fire de executie

Pentru a crea un fir de executie, se apeleaza functia CreateThread. Firul de executie va executa codul indicat printr-un parametru al functiei (o functie). Executia va porni imediat dupa apelul functiei (cu exeptia cazului in care se creaza un fir suspendat)

```
HANDLE WINAPI CreateThread(
    _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes,
    _In_ SIZE_T dwStackSize,
    _In_ LPTHREAD_START_ROUTINE lpStartAddress,
    _In_opt_ LPVOID lpParameter,
    _In_ DWORD dwCreationFlags,
    _Out_opt_ LPDWORD lpThreadId
    );
```

lpThreadAttributes	Drepturile de securitate a firului de executie
dwStackSize	Dimensiunea stivei. Deobicei este 0 pentru a fi o dimensiune standard
lpStartAddress	Pointer catre o functie asemnatoare cu cea de jos DWORD WINAPI functieFirExecutie(LPVOID lpThreadParameter) Firul de executie va executa acest cod
lpParameter	Argumentul care paote fi dat functiei identificata prin lpStartAddress
dwCreationFlags	Deobicei 0, dar poate fi creat si un fir de executie care initial este suspendat prin trimiterea flagului CREATE_SUSPENDED
lpThreadId	Id-ul firului de executie, in caz ca se doreste

In caz ca functia reuseste, va returna un handle catre acel fir. Firul principal de executie (cel care porneste din functia main) trebuie sa astepte celelalte fire de executie. Acest lucru se intampla prin intermediul functiei WaitForSingleObject care va primi ca parametrul handle-ul catre firul pe care doreste sa-l astepte.

Una din posibilele actiuni asupra unui fir de executie este suspendarea. Ca orice obiect din Windows, pentru a executa actiuni asupra firului de executie, trebuie intai obtinut un handle (in cazul in care nu avem deja unul de la functia CreateThread). Acest lucru se face cu functia OpenThread.

```
HANDLE WINAPI OpenThread(
  _In_  DWORD dwDesiredAccess,
  _In_  BOOL bInheritHandle,
  _In_  DWORD dwThreadId
);
```

dwDesiredAccess	Drepturile de securitate ce indica viitoare actiuni ce vor fi facute pe fir.ex: THREAD_QUERY_INFORMATION, THREAD_SUSPEND_RESUME, THREAD_TERMINATE
bInheritHandle	Daca acest handle poate fi mostenit de copii acestui proces
dwThreadId	Id-ul threadului pentru care se doreste handle

Suspendarea unui proces presupune suspendarea tuturor firelor. Tinand cont ca un proces reprezinta un set de resurse alocate, nu are sens notiunea de suspendare a procesului. Din acest motiv, trebuie enumerate toate firele de executie, selectat acele fire care apartin procesului care se doreste a fi suspendat, iar apoi apelat functia SuspendThread

Lucru cu fisiere mapate in memorie

Tinand cont ca fiecare proces functioneaza in cadrul memoriei sale virtuale, trebuie gasita o modalitate prin care doua procese sa poata comunica intre ele. Sistemul de operare ofera posibilitatea ca procesele sa acceseze aceiasi zona de memorie fizica, prin intermediul fisierelor mapate in memorie.

Fisierul mapat in memorie reprezinta de fapt o metoda prin care un fisier de pe disc poate fi accesat si modificat la fel ca o zona de memorie. (in loc sa apelam SetFilePointer, pur si simplu schimbam un index de la un buffer). Totusi, metoda ne permite sa alocam o zona de memorie, fara a specifica nici un fisier.

Pentru a crea un fisier mapat in memorie, trebuie apelate 2 functii:

1. CreateFileMapping pentru a aloca memorie si pentru a mapa fisierul in memoria fizica.

```
HANDLE WINAPI CreateFileMapping(
  _In_      HANDLE hFile,
  _In_opt_  LPSECURITY_ATTRIBUTES lpAttributes,
```

```
  _In_      DWORD flProtect,
  _In_      DWORD dwMaximumSizeHigh,
  _In_      DWORD dwMaximumSizeLow,
  _In_opt_  LPCTSTR lpName
);
```

hFile	Handle-ul catre fisierul de pe disk ce se doreste a fi mapat. Poate fi null
lpAttributes	Drepturi de securitate asupra obiectului mapat
flProtect	Drepturi pe paginile de memorie (ex: PAGE_READWRITE pentru scriere si citire
dwMaximumSizeHigh	Dimensiunea zonei alocate. Daca s-a specificat un handle la un fisier, este folosita dimensiunea fisierului. (doar pentru zone mai mari de 4gb)
dwMaximumSizeLow	Dimensiunea zonei alocate. Daca s-a specificat un handle la un fisier, este folosita dimensiunea fisierului
lpName	Numele obiectului. Este folosit pentru a putea fi gasit de alt proces.

Celalalt process, pentru a gasi obiectul, va trebui sa apeleze functia OpenFileMapping. Aceasta functie nu face decat sa obtina un handle catre obiectul mapat pentru a putea lucra cu el.

```
HANDLE WINAPI OpenFileMapping(
    _In_ DWORD dwDesiredAccess,
    _In_ BOOL bInheritHandle,
    _In_ LPCTSTR lpName
    );
```

dwDesiredAccess	Drepturile ce indica actiunile ce vor fi facute asupra obiectului
bInheritHandle	Daca acest handle poate fi sau nu mostenit de copii
lpName	Numele obiectului mapat. Cel dat la functia CreateFileMapping

2. Deocamdata fisierul este mapat in memoria fizica. Pentru a putea lucre cu el, fiecare process trebuie sa-l aduca in memoria virtuala. Pentru acest lucru, treubie apelata functia MapViewofFile

```
LPVOID WINAPI MapViewOfFile(
  _In_  HANDLE hFileMappingObject,
  _In_  DWORD dwDesiredAccess,
  _In_  DWORD dwFileOffsetHigh,
  _In_  DWORD dwFileOffsetLow,
  _In_  SIZE_T dwNumberOfBytesToMap
);
```

hFileMappingObject	Handle-ul obtinu prin apelarea functiei CreateFileMapping sau OpenFilemapping
dwDesiredAccess	Ce drepturi sa aiba paginile de memorie virtuala in care

	este mapat fisierul. FILE_MAP_WRITE pentru citire si scriere
dwFileOffsetHigh	In cazul maparii unui fisier, specifica de la ce offset sa inceapa maparea. Pentru fisiere sub 4gb dwFileOffsetHigh este 0
dwFileOffsetLow	In cazul maparii unui fisier, specifica de la ce offset sa inceapa maparea.
dwNumberOfBytesToMap	Numarul de bytes care sa fie mapati. Daca este 0, se va folosi dimensiunea maxima (cat s-a specificat in CreateFileMapping)

Daca functia reuseste, va returna un pointer catre zona de memorie virtuala in care a fost mapat fisierul. Mai departe, daca un proces vrea sa comunice cu un alt proces,tot ce trebuie sa faca este sa scrie date la acea adresa. Procesul celelalat, va putea sa citeasca datele de la adresa la care a fost mapat fisierul.

La sfarsit, fiecare proces trebuie sa apeleze functia CloseHandle pentru a elibera resursele. Memoria pentru fisierul mapat va fi delocata atunci cand s-a inchis ultimul handle pe fisier.

Setarea de privilegii

Privilegiile unui proces sunt salvate in tokenul procesului. Astfel, pentru a adauga sau a elimina anumite privilegii, trebuie intai acces catre token. Acest lucru se face prin apelul functiei OpenProcessToken.

```
BOOL WINAPI OpenProcessToken(
    _In_ HANDLE ProcessHandle,
    _In_ DWORD DesiredAccess,
    _Outptr_ PHANDLE TokenHandle
    );
```

ProcessHandle	Handle-ul catre procesul curent. De cele mai multe ori, se doreste ca setarea de privilegii sa fie facuta pentru procesul curent, asa ca parametrul va fi deobicei rezultatul functiei GetCurrentProcess()
DesiredAccess	drepturile care indica ce actiuni vor fi facute pe token. Ex: setare de privilegii (TOKEN_ADJUST_PRIVILEGES) sau interogare de privilegii (TOKEN_QUERY)
TokenHandle	Un pointer in care va fi returnat handle-ul catre token-ul procesului

Apoi, trebuie identificat privilegiul. Fiecare privilegiu are un id unic pe fiecare sistem. Pentru a afla id-ul trebuie apelata functia LookupPrivilegeValue

```
BOOL WINAPI LookupPrivilegeValueW(
    _In_opt_ LPCTSTR lpSystemName,
    _In_     LPCTSTR lpName,
    _Out_    PLUID   lpLuid);
```

lpSystemName	Numele sistemului de pe care se doreste sa se afle id-ul

	privilegiului. Daca este setat pe NULL, atunci va fi cautat pe sistemul curent.
lpName	Numele privilegiului
lpLuid	Pointer catre un element de tip LUID in care va fi pus id-ul privilegiului.(ex: SE_DEBUG_NAME pentru debug la sistem, SE_SHUTDOWN_NAME pentru a inchide sistemul)

Avand id-ul privilegiului si handle-ul catre token se poate seta privilegiul cu ajutorul functiei AdjustTokenPrivileges.

```
BOOL WINAPI AdjustTokenPrivileges(
  _In_       HANDLE TokenHandle,
  _In_       BOOL DisableAllPrivileges,
  _In_opt_   PTOKEN_PRIVILEGES NewState,
  _In_       DWORD BufferLength,
  _Out_opt_  PTOKEN_PRIVILEGES PreviousState,
  _Out_opt_  PDWORD ReturnLength
);
```

TokenHandle	Handle-ul catre token, obtinut prin OpenProcessToken
DisableAllPrivileges	Specifica daca vrem sa dezactivam toate privilegiile deja existente, iar lista noua de privilegii va fi data de noi. Daca se doreste doar adugare de privilegiu, atunci acest parametru va fi FALSE

NewState	Un pointer catre o structura de tip TOKEN_PRIVILEGES care indica vectorul de privilegii care vor fi setate
	
	struct TOKEN_PRIVILEGES {
	DWORD PrivilegeCount;
	LUID_AND_ATTRIBUTES Privileges[ANYSIZE_ARRAY];
	}
	
	PrivilegeCount reprezinta numarul de privilegii pe care vrem sa le adaugam/eliminam
	
	Privileges reprezinta vectorul care descrie fiecare privilegiu si daca sa fie adaugat sau eliminat. Vectorul trebuie sa aiba dimensiunea PrivilegeCount. Fiecare privilegiu este descris prin intermediul structurii LUID_AND_ATTRIBUTESS
	
	struct LUID_AND_ATTRIBUTES {
	LUID Luid;
	DWORD Attributes;
	}
	Luid reprezinta id-ul privilegiului pe sistem
	Attributes reprezinta actiunea asupra privilegiului (SE_PRIVILEGE_ENABLED sau SE_PRIVILEGE_REMOVED)

BufferLength	Dimensiunea parametrului NewState(sizeof(DWORD)+PrivilegeCount*sizeof(LUID_AND_ATTRIBUTES)

PreviousState	Un pointer catre o structura de tip TOKEN_PRIVILEGES prin care se specifica privilegiile precedente (in caz ca se doreste asa ceva)
ReturnLength	Dimensiunea structurii returnate in PreviousState

Handle-ul catre token trebuie inchis cu functia CloseHandle

Informatiile prezentate in acest fisier sunt introductive, iar o parte din ele au fost simpificate pentru a face mai usoara intelegerea conceptelor din spate. Pentru a vedea exact felul in care sunt implementate consultati cursul si cartea Windows Internals.